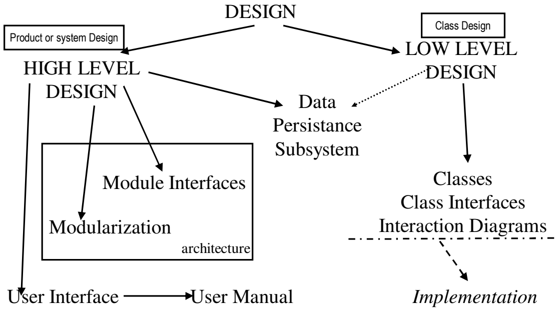
CMPT 275

Phase: Design

Map of design phase



DESIGN

Product or system Design

Class Design

HIGH LEVEL DESIGN

LOW LEVEL DESIGN

Data Persistance Subsystem

Module Interfaces

Modularization

architecture

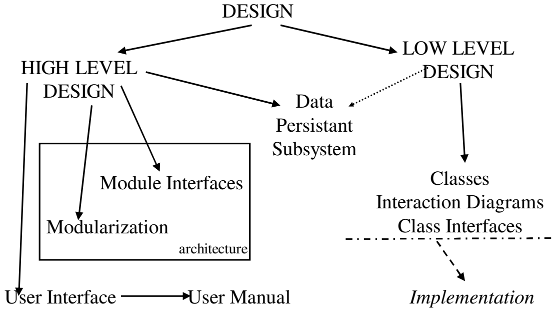
Classes
Class Interfaces
Interaction Diagrams

User Interface

User Manual

Implementation

Map of design phase



DESIGN

HIGH LEVEL DESIGN

LOW LEVEL DESIGN

Data Persistant Subsystem

Module Interfaces

Modularization

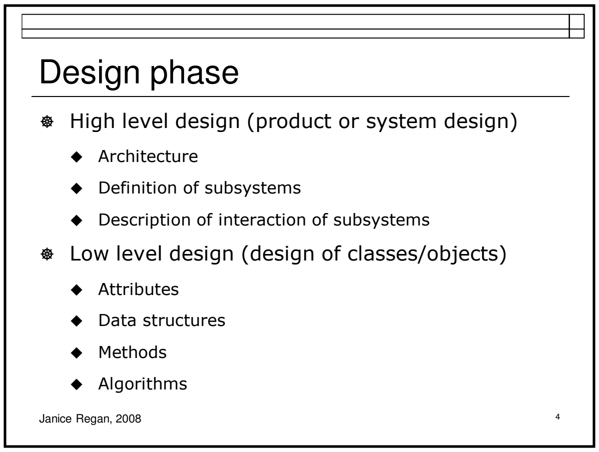
architecture

Classes
Interaction Diagrams
Class Interfaces

User Interface

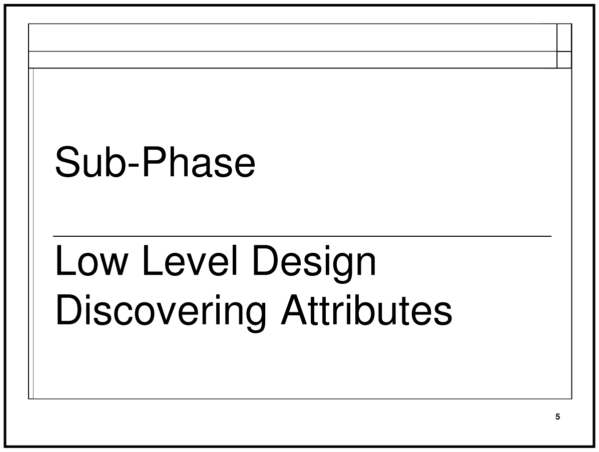
User Manual

Implementation



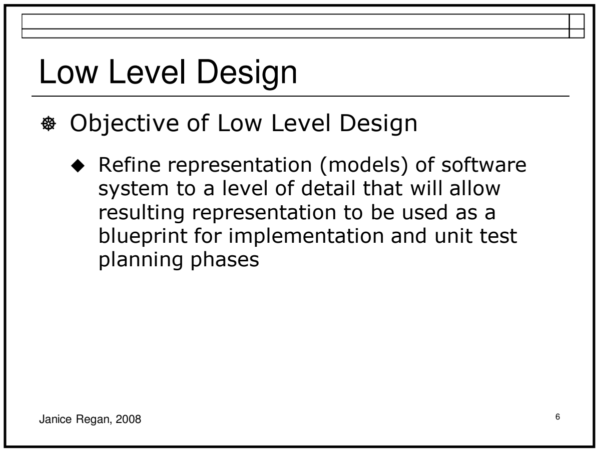
Design phase

- High level design (product or system design)
  - Architecture
  - Definition of subsystems
  - Description of interaction of subsystems
- Low level design (design of classes/objects)
  - Attributes
  - Data structures
  - Methods
  - Algorithms



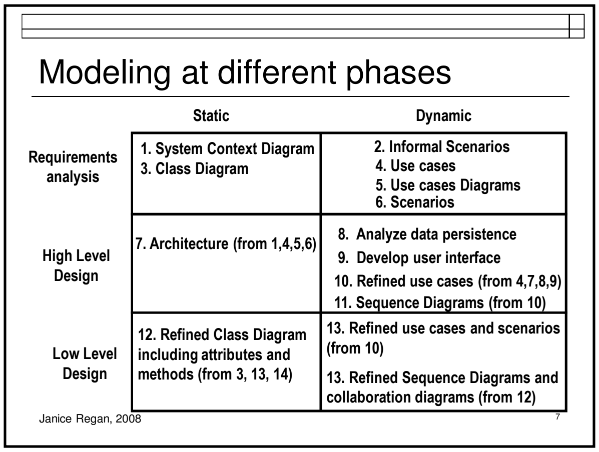
Sub-Phase

Low Level Design Discovering Attributes



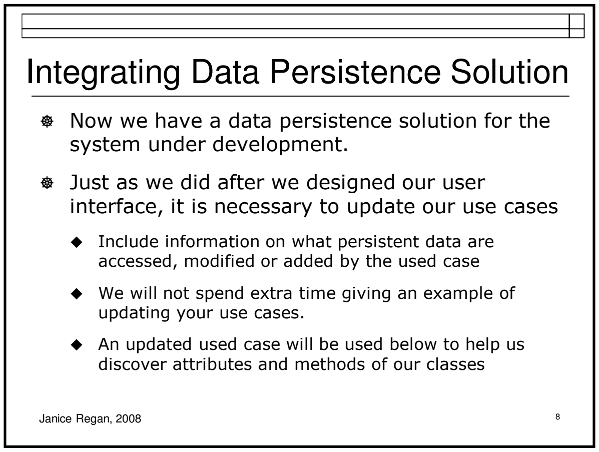
Low Level Design

- Objective of Low Level Design
  - Refine representation (models) of software system to a level of detail that will allow resulting representation to be used as a blueprint for implementation and unit test planning phases



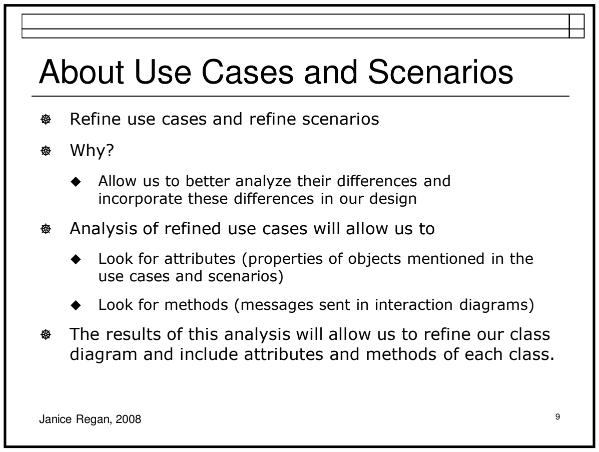
Modeling at different phases

	Static	Dynamic
Requirements analysis	1. System Context Diagram 3. Class Diagram	2. Informal Scenarios 4. Use cases 5. Use cases Diagrams 6. Scenarios
High Level Design	7. Architecture (from 1,4,5,6)	8. Analyze data persistence 9. Develop user interface 10. Refined use cases (from 4,7,8,9) 11. Sequence Diagrams (from 10)
Low Level Design	12. Refined Class Diagram including attributes and methods (from 3, 13, 14)	13. Refined use cases and scenarios (from 10) 13. Refined Sequence Diagrams and collaboration diagrams (from 12)



Integrating Data Persistence Solution

- Now we have a data persistence solution for the system under development.
- Just as we did after we designed our user interface, it is necessary to update our use cases
  - Include information on what persistent data are accessed, modified or added by the used case
  - We will not spend extra time giving an example of updating your use cases.
  - An updated used case will be used below to help us discover attributes and methods of our classes



About Use Cases and Scenarios

- Refine use cases and refine scenarios
- Why?
  - Allow us to better analyze their differences and incorporate these differences in our design
- Analysis of refined use cases will allow us to
  - Look for attributes (properties of objects mentioned in the use cases and scenarios)
  - Look for methods (messages sent in interaction diagrams)
- The results of this analysis will allow us to refine our class diagram and include attributes and methods of each class.

Let’s get started!

- How to transform our Class Diagram (created in R.A.) into refined classes that can be used as blueprints?
  - Look at a sample refined use case
    - Discover attributes of classes, by considering possible scenarios associated with the use case
    - Update the attributes on your class
    - Consider sequence diagrams and/or colloboration diagrams associated with the use case to help identify the methods needed for each class
  - Consider each use case in the above manner to assure that all attributes and methods have been identified

CheckInResource Use Case

A. Use Case Name : CheckInResource (#7)

B. Initiating actor: Librarian

C. Preconditions:

- Librarian is a valid librarian
- LMS is ready to go (DB has been populated, network is up, and LMS has been initialized)
- LMS screen with Check menu is displayed.

CheckInResource Use Case

D. Main flow of events:

1. The use case starts when **Librarian** chooses CheckInResource option from the LMS screen by selecting the ***In*** command option under the ***Check*** menu.
2. A window representing a Check In Form is then displayed. The Librarian types the *Dewey call number for the resource* in the appropriate text field then presses the "Accept" button to commit the entry.

CheckInResource Use Case

D. Main flow of events (cont):

3. If the Dewey call number for the resource has been entered successfully and it is valid (i.e., it does refer to a resource of this library), *the resource info and the borrowing patron info* are retrieved from the Database and displayed on the Check In screen.
4. The DetermineOverdueCharge use case is initiated.
5. LMS verifies that there is no outstanding request for this resource.

CheckInResource Use Case

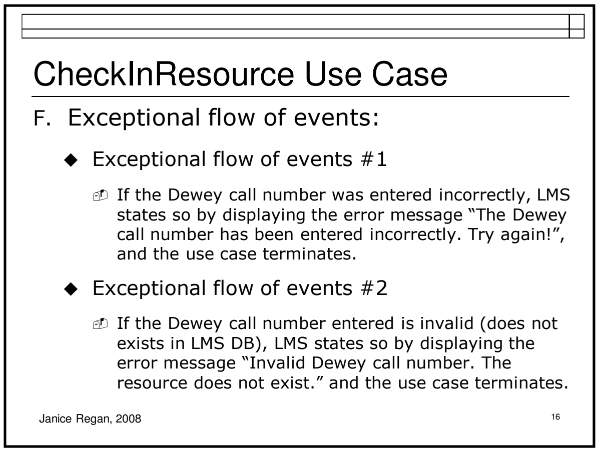
D. Main flow of events: (cont'd)

- If there are no outstanding request for this resource, LMS changes the *status* of the resource to "reshelve" and cancels its *"due date"* and *"date of loan"* and updates its *"date of return*" to today. It updates the resource and patron records in the Database.
- Finally, LMS updates the screen showing the newly checked-in resource along with the updated dates.
- The use case terminates when Librarian indicates that she/he is done by pressing the "Done" button.

CheckInResource Use Case

E. Postconditions:

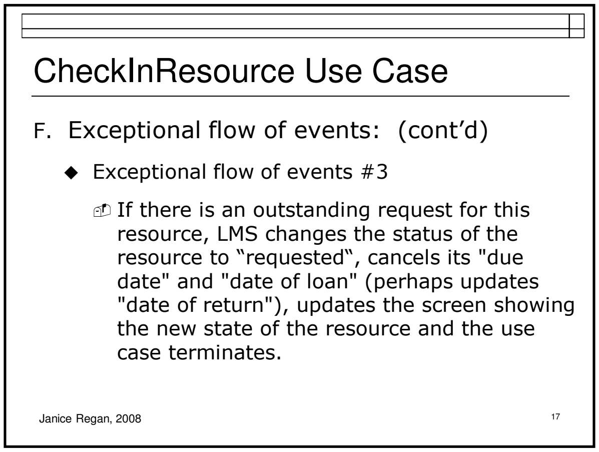
- ◆ If the use case was successfully executed:
  - Patron object/record updated to reflect the newly checked in resource.
  - Resource object/record updated to reflect its checked in status and dates.
- ◆ Back to initial LMS screen.



CheckInResource Use Case

F. Exceptional flow of events:

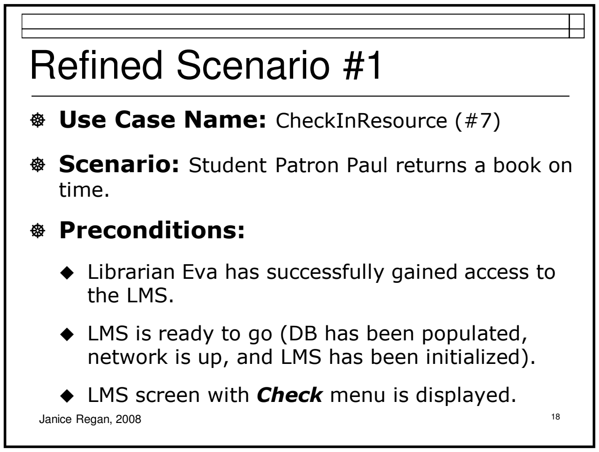
- Exceptional flow of events #1
  - If the Dewey call number was entered incorrectly, LMS states so by displaying the error message "The Dewey call number has been entered incorrectly. Try again!", and the use case terminates.
- Exceptional flow of events #2
  - If the Dewey call number entered is invalid (does not exists in LMS DB), LMS states so by displaying the error message "Invalid Dewey call number. The resource does not exist." and the use case terminates.



CheckInResource Use Case

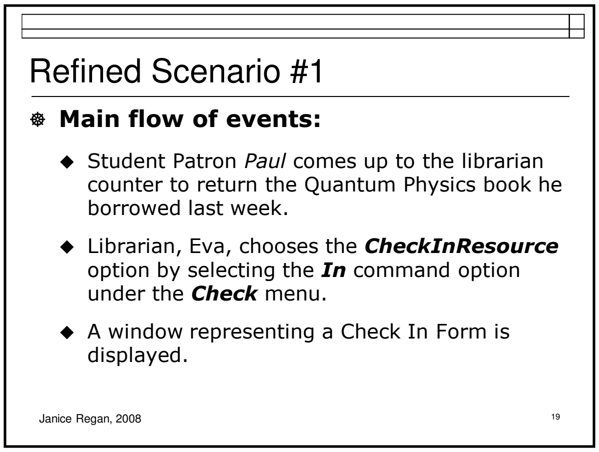
F. Exceptional flow of events: (cont'd)

- Exceptional flow of events #3
  - If there is an outstanding request for this resource, LMS changes the status of the resource to "requested", cancels its "due date" and "date of loan" (perhaps updates "date of return"), updates the screen showing the new state of the resource and the use case terminates.



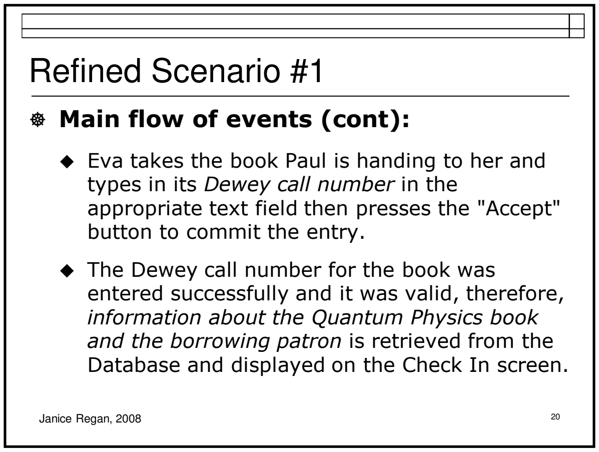
Refined Scenario #1

- **Use Case Name:** CheckInResource (#7)
- **Scenario:** Student Patron Paul returns a book on time.
- **Preconditions:**
  - Librarian Eva has successfully gained access to the LMS.
  - LMS is ready to go (DB has been populated, network is up, and LMS has been initialized).
  - LMS screen with ***Check*** menu is displayed.



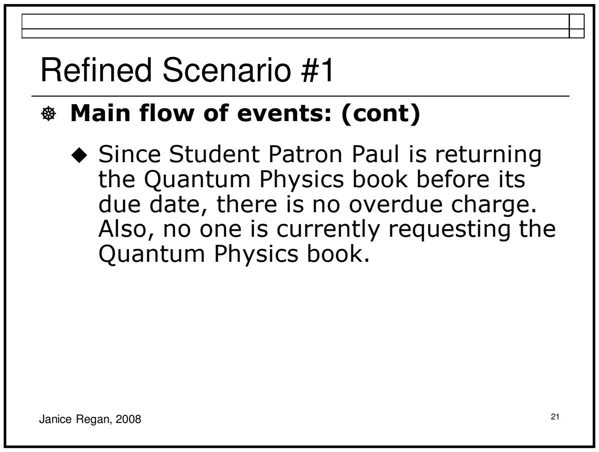
Refined Scenario #1

- **Main flow of events:**
  - Student Patron *Paul* comes up to the librarian counter to return the Quantum Physics book he borrowed last week.
  - Librarian, Eva, chooses the ***CheckInResource*** option by selecting the ***In*** command option under the ***Check*** menu.
  - A window representing a Check In Form is displayed.



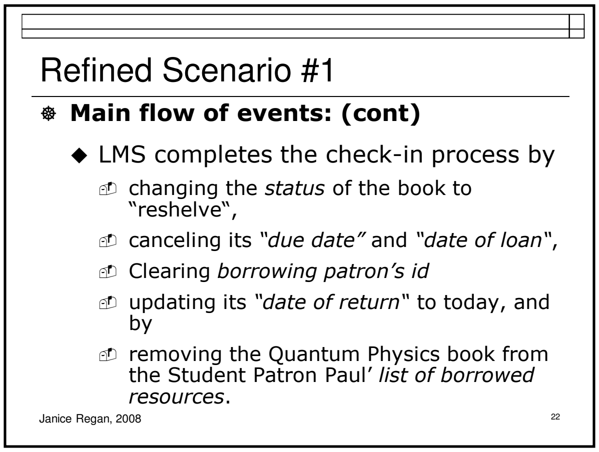
Refined Scenario #1

- **Main flow of events (cont):**
  - Eva takes the book Paul is handing to her and types in its *Dewey call number* in the appropriate text field then presses the "Accept" button to commit the entry.
  - The Dewey call number for the book was entered successfully and it was valid, therefore, *information about the Quantum Physics book and the borrowing patron* is retrieved from the Database and displayed on the Check In screen.



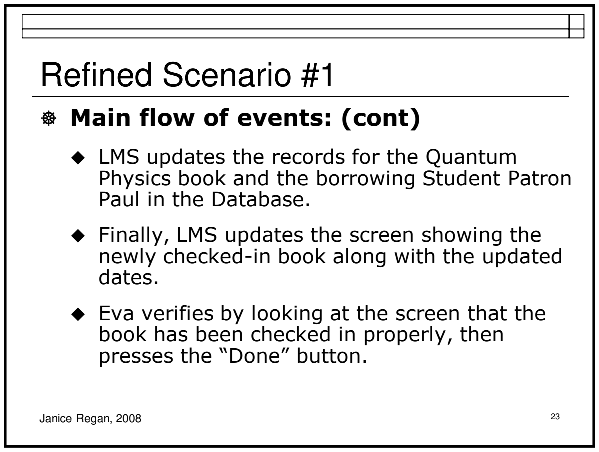
Refined Scenario #1

- **Main flow of events: (cont)**
  - Since Student Patron Paul is returning the Quantum Physics book before its due date, there is no overdue charge. Also, no one is currently requesting the Quantum Physics book.



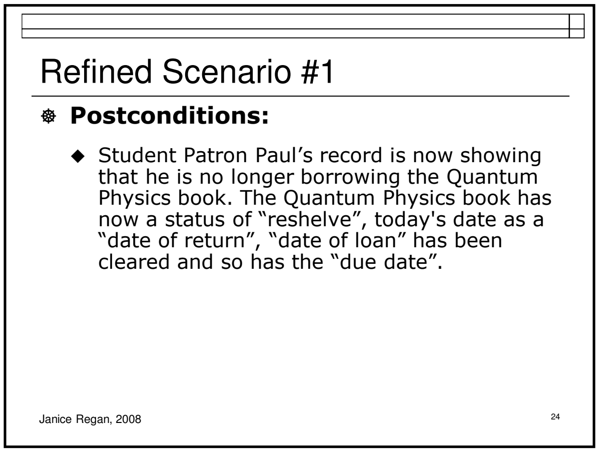
Refined Scenario #1

- **Main flow of events: (cont)**
  - LMS completes the check-in process by
    - changing the *status* of the book to “reshelve‟,
    - canceling its *“due date”* and *“date of loan*‟,
    - Clearing *borrowing patron’s id*
    - updating its *“date of return*‟ to today, and by
    - removing the Quantum Physics book from the Student Patron Paul’ *list of borrowed resources*.



Refined Scenario #1

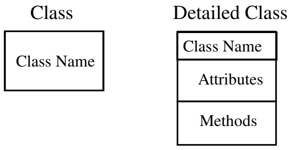
- **Main flow of events: (cont)**
  - LMS updates the records for the Quantum Physics book and the borrowing Student Patron Paul in the Database.
  - Finally, LMS updates the screen showing the newly checked-in book along with the updated dates.
  - Eva verifies by looking at the screen that the book has been checked in properly, then presses the “Done” button.



Refined Scenario #1

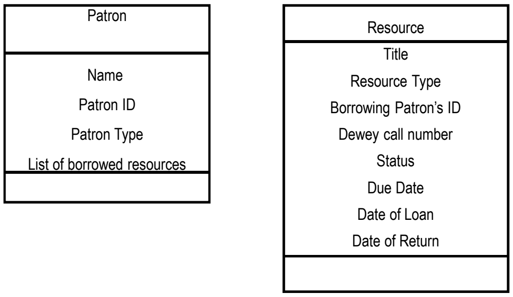
- **Postconditions:**
  - Student Patron Paul’s record is now showing that he is no longer borrowing the Quantum Physics book. The Quantum Physics book has now a status of “reshelve”, today's date as a “date of return”, “date of loan” has been cleared and so has the “due date”.

UML: Class Diagrams



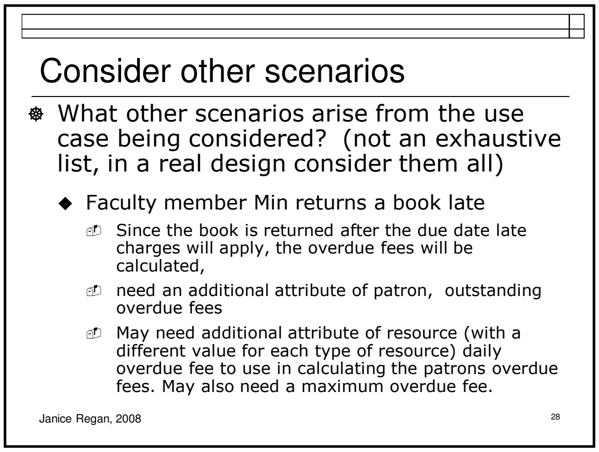
- In requirements analysis we used the simple version of a single box to represent a class
- In design we will use the detailed form of the class
  - Need to identify attributes and methods of each class as part of the design process. From our analysis of the scenario we have a first estimate of the attributes of our first 2 classes

Patron and Resource class attributes



Consider other scenarios

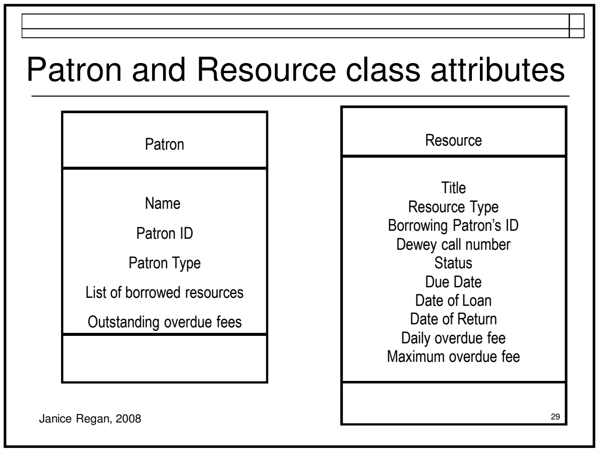
- What other scenarios arise from the use case being considered? (not an exhaustive list, in a real design consider them all)
  - Faculty member Sue returns a book on time rather than student Paul
    - Length of time the resource (book) is borrowed for changes: This effects only the determination of the value of the due date determined when the book is checked out (not part of the use case being considered), it does not require any additional attributes.



Consider other scenarios

- What other scenarios arise from the use case being considered? (not an exhaustive list, in a real design consider them all)
  - Faculty member Min returns a book late
    - Since the book is returned after the due date late charges will apply, the overdue fees will be calculated,
    - need an additional attribute of patron, outstanding overdue fees
    - May need additional attribute of resource (with a different value for each type of resource) daily overdue fee to use in calculating the patrons overdue fees. May also need a maximum overdue fee.

Janice Regan, 2008 28



Patron and Resource class attributes

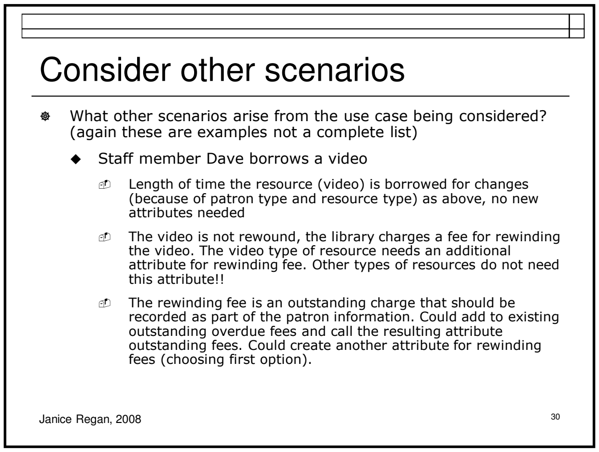
Patron

Name
Patron ID
Patron Type
List of borrowed resources
Outstanding overdue fees

Resource

Title
Resource Type
Borrowing Patron's ID
Dewey call number
Status
Due Date
Date of Loan
Date of Return
Daily overdue fee
Maximum overdue fee

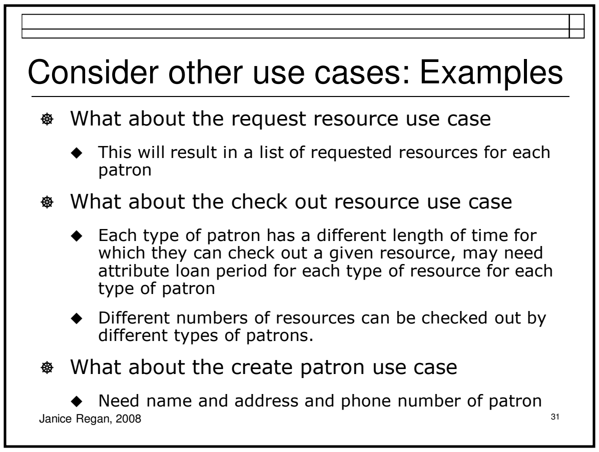
Janice Regan, 2008 29



Consider other scenarios

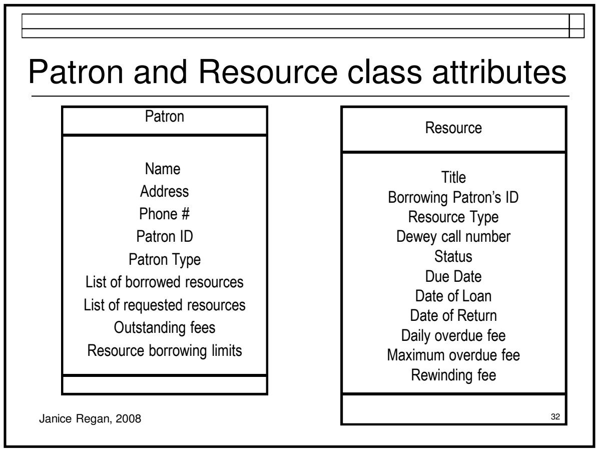
- What other scenarios arise from the use case being considered? (again these are examples not a complete list)
  - Staff member Dave borrows a video
    - Length of time the resource (video) is borrowed for changes (because of patron type and resource type) as above, no new attributes needed
    - The video is not rewound, the library charges a fee for rewinding the video. The video type of resource needs an additional attribute for rewinding fee. Other types of resources do not need this attribute!!
    - The rewinding fee is an outstanding charge that should be recorded as part of the patron information. Could add to existing outstanding overdue fees and call the resulting attribute outstanding fees. Could create another attribute for rewinding fees (choosing first option).

Janice Regan, 2008 30



Consider other use cases: Examples

- What about the request resource use case
  - This will result in a list of requested resources for each patron
- What about the check out resource use case
  - Each type of patron has a different length of time for which they can check out a given resource, may need attribute loan period for each type of resource for each type of patron
  - Different numbers of resources can be checked out by different types of patrons.
- What about the create patron use case
  - Need name and address and phone number of patron



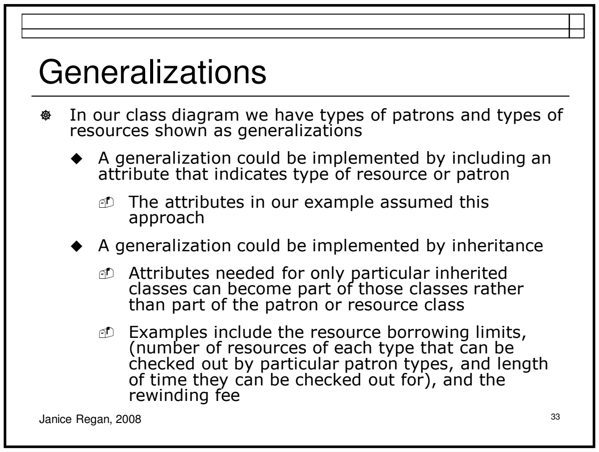
Patron and Resource class attributes

Patron

Name
Address
Phone #
Patron ID
Patron Type
List of borrowed resources
List of requested resources
Outstanding fees
Resource borrowing limits

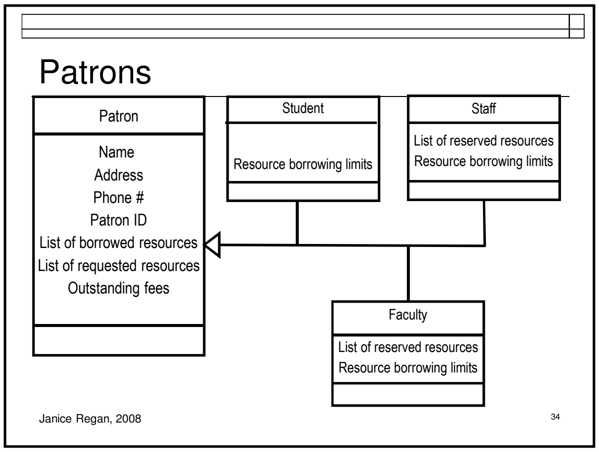
Resource

Title
Borrowing Patron's ID
Resource Type
Dewey call number
Status
Due Date
Date of Loan
Date of Return
Daily overdue fee
Maximum overdue fee
Rewinding fee



Generalizations

- In our class diagram we have types of patrons and types of resources shown as generalizations
  - A generalization could be implemented by including an attribute that indicates type of resource or patron
    - The attributes in our example assumed this approach
  - A generalization could be implemented by inheritance
    - Attributes needed for only particular inherited classes can become part of those classes rather than part of the patron or resource class
    - Examples include the resource borrowing limits, (number of resources of each type that can be checked out by particular patron types, and length of time they can be checked out for), and the rewinding fee

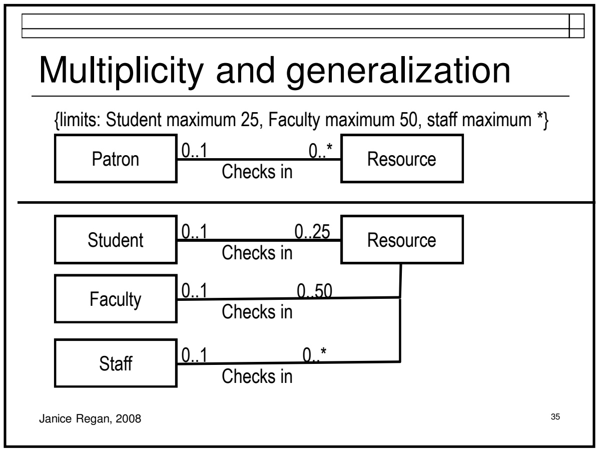


Patrons

Patron
Name
Address
Phone #
Patron ID
List of borrowed resources
List of requested resources
Outstanding fees
Student
Resource borrowing limits
Staff
List of reserved resources
Resource borrowing limits
Faculty
List of reserved resources
Resource borrowing limits

Janice Regan, 2008

34

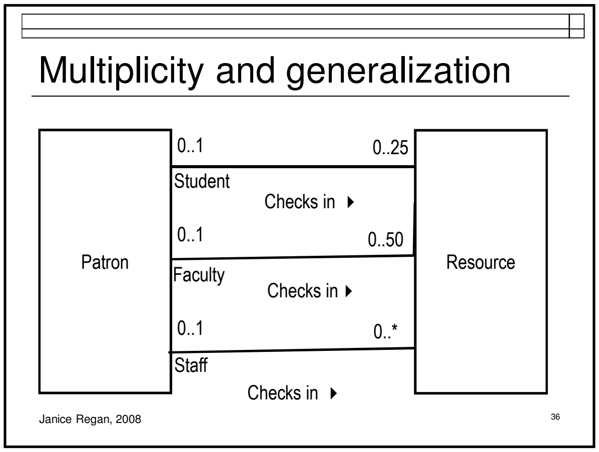


Multiplicity and generalization

{limits: Student maximum 25, Faculty maximum 50, staff maximum *}

Janice Regan, 2008

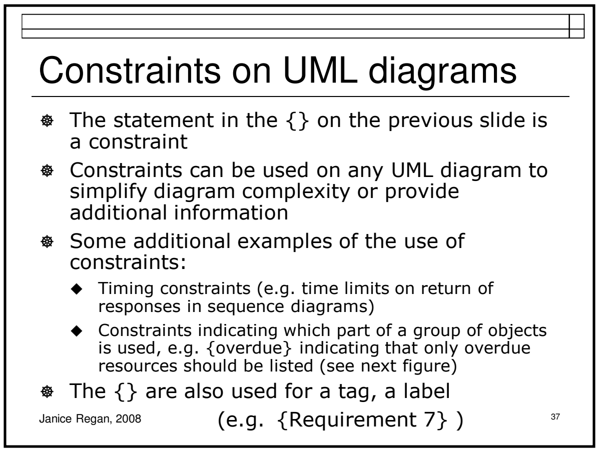
35



Multiplicity and generalization

Janice Regan, 2008

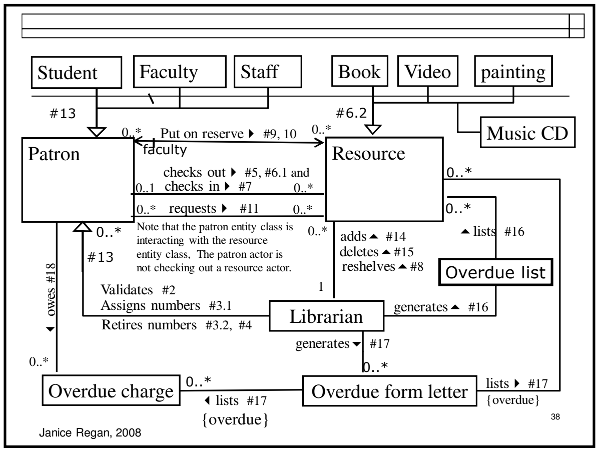
36



Constraints on UML diagrams

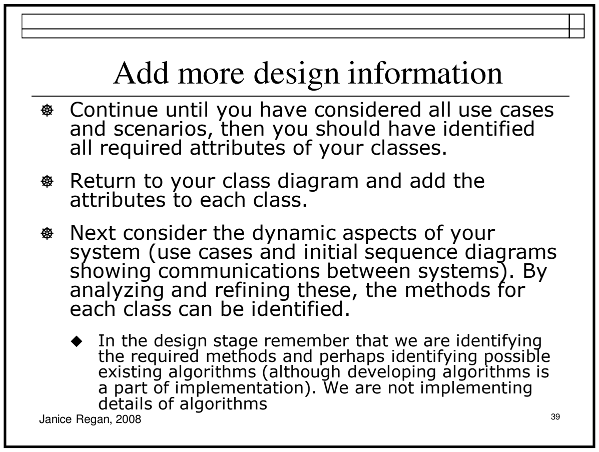
- The statement in the {} on the previous slide is a constraint
- Constraints can be used on any UML diagram to simplify diagram complexity or provide additional information
- Some additional examples of the use of constraints:
  - Timing constraints (e.g. time limits on return of responses in sequence diagrams)
  - Constraints indicating which part of a group of objects is used, e.g. {overdue} indicating that only overdue resources should be listed (see next figure)
- The {} are also used for a tag, a label (e.g. {Requirement 7})

Janice Regan, 2008 37



Note that the patron entity class is interacting with the resource entity class, The patron actor is not checking out a resource actor.

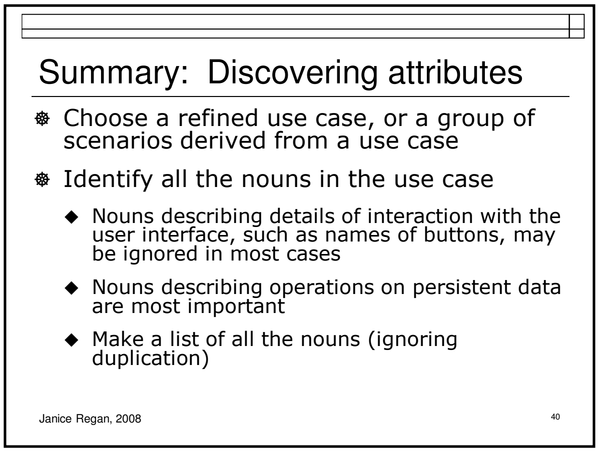
Janice Regan, 2008 38



Add more design information

- Continue until you have considered all use cases and scenarios, then you should have identified all required attributes of your classes.
- Return to your class diagram and add the attributes to each class.
- Next consider the dynamic aspects of your system (use cases and initial sequence diagrams showing communications between systems). By analyzing and refining these, the methods for each class can be identified.
  - In the design stage remember that we are identifying the required methods and perhaps identifying possible existing algorithms (although developing algorithms is a part of implementation). We are not implementing details of algorithms

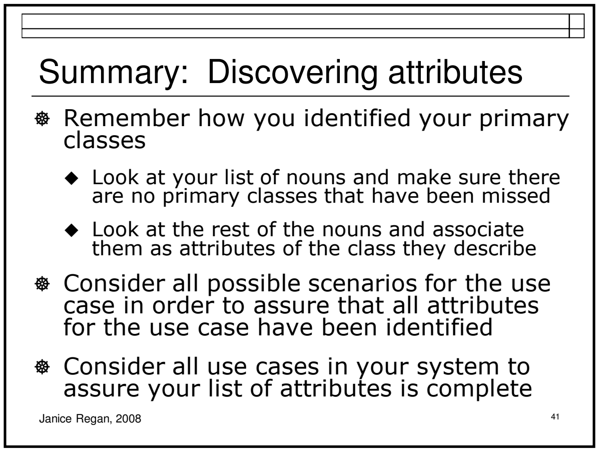
Janice Regan, 2008 39



Summary: Discovering attributes

- Choose a refined use case, or a group of scenarios derived from a use case
- Identify all the nouns in the use case
  - Nouns describing details of interaction with the user interface, such as names of buttons, may be ignored in most cases
  - Nouns describing operations on persistent data are most important
  - Make a list of all the nouns (ignoring duplication)

Janice Regan, 2008 40



Summary: Discovering attributes

- Remember how you identified your primary classes
  - Look at your list of nouns and make sure there are no primary classes that have been missed
  - Look at the rest of the nouns and associate them as attributes of the class they describe
- Consider all possible scenarios for the use case in order to assure that all attributes for the use case have been identified
- Consider all use cases in your system to assure your list of attributes is complete

Janice Regan, 2008 41